VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

IN THE TRACES OF LEOŠ JANÁČEK - CONVERSION OF SPEECH TO MUSIC

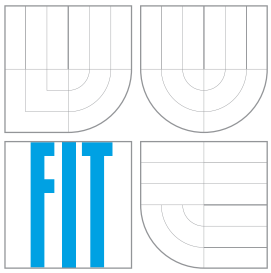
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR MARCINIAK

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VE STOPÁCH LEOŠE JANÁČKA - PŘEVOD ŘEČI NA HUDBU

IN THE TRACES OF LEOŠ JANÁČEK - CONVERSION OF SPEECH TO MUSIC

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR MARCINIAK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. JAN ČERNOCKÝ

BRNO 2010

Abstrakt

Tato bakalářská práce popisuje vývoj aplikace pro převod řeči z nahrávky ve formátu WAV na hudbu uloženou ve formátu MIDI. V úvodní části je čtenář uveden do problematiky. Následuje popis teoretických základů zpracování řeči a následného generování hudby. Dále jsou diskutovány počáteční experimenty, jako generování základní melodie, průměrování tónů, detekce slabik atp., za účelem určení, které z těchto technik mají pozitivní vliv na poslouchatelnost vytvořené hudby, a proto by měly být ve výsledné aplikaci implementovány. Následně jsou definována základní kritéria krásy z hlediska generování hudby a jsou diskutovány různé skladatelské techniky, jako např. inverze tónů nebo změna tempa. Následuje popis implementace a vyhodnocení provedených testů. V závěrečné části je celá práce zhodnocena a je zde i krátké zamyšlení nad možnými dalšími směry vývoje tohoto systému. V příloze je možné najít uživatelský manuál k aplikaci a dále také seznam nástrojů použitých pro implementaci.

Abstract

The aim of this bachelor thesis is to develop an application, which will automatically convert speech recording in WAV format to speech-melody-based music in MIDI format. At first, the problem is analyzed and the theoretical background is described. Basics of music generation from speech are introduced. Initial experiments like creation of the elementary melody, averaging of tones, syllables detection, etc. are discussed in order to establish, which of these techniques have a positive impact on the resulting music and therefore should be implemented in the resulting application. Basic criteria of beauty in music generation needed to be defined and different compositional techniques such as inversion of notes or tempo changes were investigated. Further, the implementation is described and user testing is evaluated. The conclusions are drawn and future directions of development are discussed. The user manual for the application as well as a „cook book“ listing tools used in the application development can be found in the Appendix.

Klíčová slova

konverze řeči na hudbu, rozpoznávání řeči, rozpoznávání fonémů, automatické generování hudby, MIDI

Keywords

speech to music conversion, speech recognition, phoneme recognition, automatic music generation, MIDI

Citace

Petr Marciniak: In the Traces of Leoš Janáček - Conversion of Speech to Music, bakalářská práce, Brno, FIT VUT v Brně, 2010

In the Traces of Leoš Janáček - Conversion of Speech to Music

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Dr. Ing. Jana Černockého

.....
Petr Marciniak
May 18, 2010

Poděkování

Touto cestou bych velmi rád poděkoval především vedoucímu práce doc. Dr. Ing. Janu Černockému za vstřícnost, ochotu pomoci, podněty a cenné rady v průběhu celého vývoje této bakalářské práce. Chtěl bych zde také poděkovat panu Mgr. Ing. MgA. Danu Dlouhému, Ph.D. za inspiraci při definování estetických kritérií z hlediska hudby a za uvedení do problematiky moderních skladatelských technik. V neposlední řadě děkuji rovněž panu Ing. Luboru Přikrylovi za nastínění možného dalšího vývoje systému a za zprostředkování konzultace s panem Dlouhým.

Contents

1 Introduction **3**
- 1.1 Description of the problem . . . 3
- 1.2 Motivation . . . 3
- 1.3 Analysis of the problem . . . 3

2 Theoretical background and terminology **4**
- 2.1 WAV (Waveform Audio File Format) . . . 4
- 2.2 Fundamental frequency . . . 4
- 2.3 Conversion of fundamental frequency to musical notes . . . 4
- 2.4 Power . . . 5
- 2.5 Phoneme Recognizer . . . 5
- 2.6 MIDI (Musical Instrument Digital Interface) . . . 6

3 Basic music generation from speech **7**
- 3.1 Requirements for input data . . . 7
- 3.2 Selection of test data . . . 7
- 3.3 Information important for basic music generation from speech . . . 7
- 3.4 Initial experiments . . . 8
  - 3.4.1 Elementary melody . . . 8
  - 3.4.2 Averaging . . . 8
  - 3.4.3 Cutting off . . . 9
  - 3.4.4 Merging . . . 10
- 3.5 Use of phoneme recognizer . . . 10
- 3.6 Syllables detection . . . 10

4 Criteria of beauty in music generation **13**
- 4.1 Motivation . . . 13
- 4.2 Dynamics . . . 13
- 4.3 Volume calculation . . . 13
- 4.4 Musical scales . . . 14
  - 4.4.1 Key note . . . 14
  - 4.4.2 Major scales . . . 14
  - 4.4.3 Minor scales . . . 14
  - 4.4.4 Cutting off . . . 15
  - 4.4.5 Adding threshold . . . 15
- 4.5 Chords . . . 15
- 4.6 Compositional Methods . . . 16
  - 4.6.1 Inversion of notes . . . 16

4.6.2 Changes of tempo . . . 16
4.6.3 Adding a bass line . . . 16
4.7 Use of more than one method/aspect at the same time . . . 16
4.8 Different musical instruments . . . 16

5 Implementation **17**
5.1 What user wants to do . . . 17
5.2 Obtaining the basic information from given WAV file . . . 17
5.3 Averaging . . . 17
5.4 Syllables detection . . . 18
5.5 Scale key note detection . . . 19
5.6 Determining if the tone fits the scale . . . 19
5.7 Creation of the MIDI File . . . 19
5.8 Waveform display . . . 22

6 Results and evaluation **23**
6.1 Examples and user Testing . . . 23
6.2 Testing results and evaluation . . . 24

7 Conclusion and future work **25**
7.1 Summary . . . 25
7.2 Contribution of the thesis . . . 25
7.3 Possible further use of the application . . . 25
7.4 Future work . . . 26
7.4.1 Music composition . . . 26
7.4.2 Resemblance to the input speech . . . 26

A Cook book **29**
A.1 Programming language . . . 29
A.2 Used tools . . . 29
A.2.1 Tkinter + Snack . . . 29
A.2.2 Phoneme recognizer based on long temporal context . . . 29
A.2.3 Python Midi Package . . . 29

B Manual for the application **30**
B.1 Requirements . . . 30
B.2 User specified options . . . 30

C CD contents **32**

Chapter 1

Introduction

1.1 Description of the problem

Czech composer Leoš Janáček used to write notations on his sleeve cuffs while listening to women of Líšeň (village close to Brno) speaking their traditional dialect on the market. He used these short pieces of melody in his world-famous operas. Our task was to develop a simple application for conversion of speech to music. The application should extract the speech melody from given WAV file containing the input sample of speech. The output should be a playable MIDI file containing the final result of the conversion of speech to music.

1.2 Motivation

This problem seemed very interesting for an amateur musician like me. There were few challenges, that needed to be faced. It was important to define, what information was needed to be extracted from speech sample for the conversion of speech to music. It was also necessary to define the criteria of beauty in music generation, so the output would sound „good“. These aspects were also discussed with Mgr. Ing. MgA. Dan Dlouhý, Ph.D. of Janáček Academy of Music and Performing Arts in Brno. Together with my supervisor and Mr. Dlouhý we wanted to compare the results of the computer conversion of speech to music and work of composer Leoš Janáček. We also wanted to check, how people like the computer generated music, when the human speech is taken as a source of the inspiration.

1.3 Analysis of the problem

At first the input WAV file containing the speech should be divided into 10 ms frames and for each frame the fundamental frequency and power of the sound should be extracted. This frequency (called pitch) will be further used to determine the musical note of the sample. The power should be useful for making the output melody more beautiful (forte, piano, etc.). The phoneme recognizer will be used to determine the syllables and pauses between words, so the notes and rhythm of the output melody are more similar to the melody of the input speech. The output should be a MIDI file containing the resulting melody.

Chapter 2

Theoretical background and terminology

2.1 WAV (Waveform Audio File Format)

Wrapper file format that can incorporate an audio bit stream with other data chunks created by Microsoft and introduced with Windows 3.1. The default bit stream encoding is the Microsoft Pulse Code Modulation (LPCM) format [14].

2.2 Fundamental frequency

The fundamental frequency (also called pitch) of sound is the essential information, that needs to be extracted from the speech sample. Therefore the speech is divided into 10 ms frames and the pitch is detected for each frame using the AMDF (Average Magnitude Difference Function) method. The basic algorithm is described in the Figure 2.1. Function implementing this method can be found in the Snack library (see Appendix A.2.1) and was further used in the application.

For each frame k, the short-term difference function AMDF is defined as follows:

$$AMDF_n(j) = \frac{1}{N}\sum_{i=1}^{N}|x_n(i) - x_n(i+j)|, 1 \leq j \leq MAXLAG \tag{2.1}$$

Where $MAXLAG$ is the maximum number of AMDF values generated in each frame. The difference function is expected to have a strong local minimum if the lag j is equal to or very close to the fundamental period. For each frame, the lag for which the AMDF is a global minimum is a strong candidate for the pitch period of that frame [21].

2.3 Conversion of fundamental frequency to musical notes

The resulting note of each frame is easy to determine from its fundamental frequency. The Concert A (also called A440 or Concert pitch) is the musical note A above the middle C vibrating at 440 times per second (440 Hz). It has been universally accepted as the pitch to which all instruments should be tuned. It ensures that when instruments play together, they will all be in tune with one another [15]. The result note of each frame is represented as:

$$12\log_2 \frac{f}{440} \tag{2.2}$$

Where 12 is a number of semitones in one octave and 440 Hz is the frequency of the Concert A. The resulting value is rounded. The value represents a distance (how many semitones above or below) from the Concert A is the resulting note.

2.4 Power

A list of power values for each 10 ms frame of the input sample is obtained. Energy values for each frame of input speech are calculated first according to the standard figure for the short-time energy calculation[22]. Function implementing this method can be found in the Snack library (see Appendix A.2.1) and was further used in the application.

$$E = \frac{1}{l_{ram}} \sum_{n=0}^{l_{ram}-1} x^2[n] \tag{2.3}$$

The list of power values is calculated according to the standard power calculation.

$$P_n = \frac{E_n}{t} \tag{2.4}$$

Where E stands for energy of sound in the frame and t stands for the time length of the frame (10 ms in our case).

2.5 Phoneme Recognizer

The Brno University of Technology phone recognizer [20] is based on hybrid ANN/HMM approach, where artificial neural networks (ANN) are used to estimate posterior probabilities of phones. The feature extraction uses Mel filter bank energies which are obtained in the conventional way. Temporal evolutions of critical band spectral densities (310ms around the current frame) are taken. The temporal context is split into left and right context. This allows for more precise modeling of the whole trajectory while limiting the size of the model (number of weights in the NN) and reducing the amount of necessary training data. Both parts are processed by discrete cosine transform (DCT) to de-correlate and reduce dimensionality. Two NNs are trained to produce the phoneme posterior probabilities for both context parts. Third NN functions as a merger and produces the final set of posterior probabilities. A classical Viterbi decoder produces final string or lattice of phones. In [19], we have shown that this system outperforms phone recognizers with GMM/HMM modeling.

The version of phone recognizer to be used in this project works with Czech phone set, the networks were trained on Czech SpeechDat-East database [2]. The system was successfully used for many tasks (language recognition, voice activity detection, on-line keyword spotting, and others) by BUT, and others (MIT, etc.).

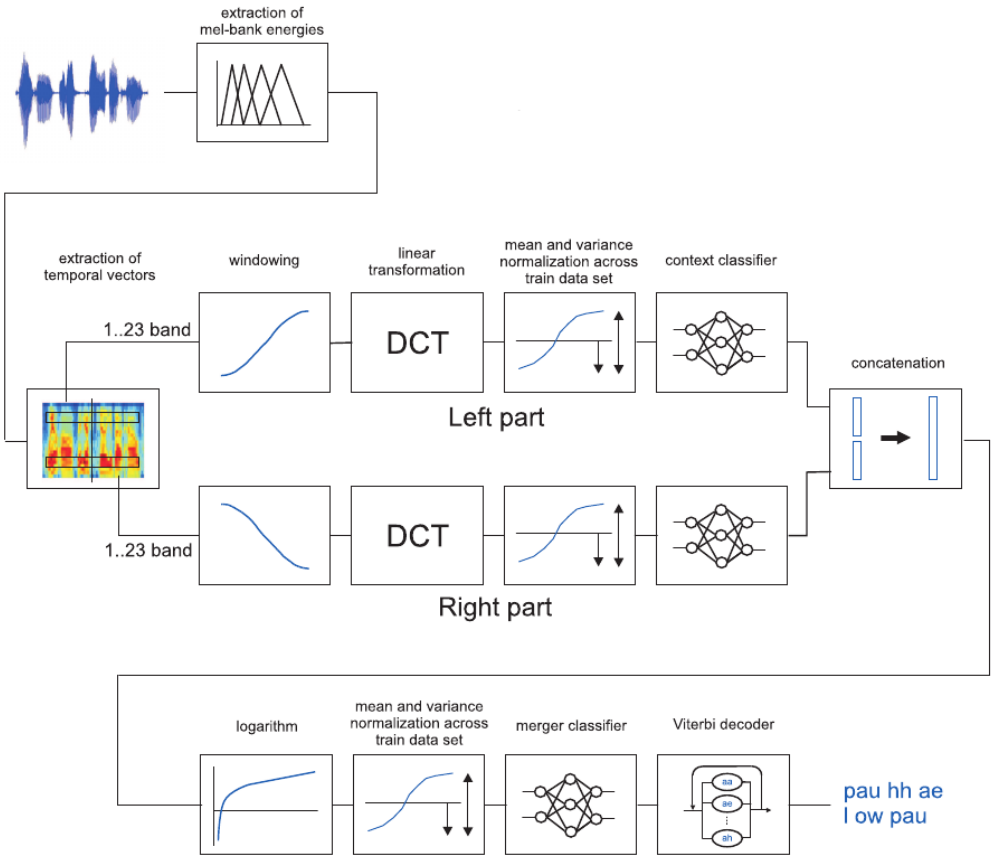


Figure 2.1: Block diagram of the Split Temporal Context system [18].

2.6 MIDI (Musical Instrument Digital Interface)

MIDI is an industry-standard protocol defined in 1982 that enables electronic musical instruments such as keyboard controllers, computers, and other electronic equipment to communicate, control, and synchronize with each other. MIDI allows computers, synthesizers, MIDI controllers, sound cards, samplers and drum machines to control one another, and to exchange system data. MIDI does not transmit an audio signal, but simply event messages [4].

The most important events messages and their attributes for this application can be found below.

Note on message Starts the playback of the note.

Note off message Ends the playback of the note.

Velocity May stand for different functionality. Velocity was used to describe the volume of the note in the resulting application. The range of possible Velocity values is 0 to 127 [6].

Chapter 3

Basic music generation from speech

3.1 Requirements for input data

The criteria for the input files were defined to simplify the implementation of the application. Therefore input files should be single-channel 16 kHz sample rate WAV files for the wave analysis (obtaining pitch and power information). However, those 16 kHz files should be re-sampled to 8 kHz in order to fit the phoneme recognizer requirements. The used phoneme recognizer was described in the Section 2.5. It is also important that the input speech signal is as clean as possible (possibly no other sounds than the speech). The possible background noise would negatively affect results of further processing of the signal.

3.2 Selection of test data

Test samples of dialect speeches were searched to make the final results of the speech to music conversion more comparable with Leoš Janáčeks work. Unfortunately there were no samples of the original dialect spoken in Líšeň found.

Samples of dialects from different regions of Poland were found [5]. Ten files were chosen. The criteria was to choose samples with the least noise in the recording. There were both male and female speech samples chosen. Duration of all of those files was cut to approximately one minute, which was considered to be enough for testing purposes. These samples were further converted from original stereo mp3 to WAV format meeting all requirements mentioned in the Section 3.1. These samples can be found on the attached CD (see Appendix C).

3.3 Information important for basic music generation from speech

It is not necessary to determine the exact frequency in case of this project. Values, that we get by using the equation 2.2 mentioned above, are satisfactory for further creation of MIDI file.

3.4 Initial experiments

All the steps mentioned further were very important to take in order to determine, which aspects of the input speech data are the most important for the music generation. These steps were also very helpful, when choosing the algorithms giving the most desirable results for further implementation.

Note: The following figures containing MIDI visualizations are results of processing of the same speech sample 01.wav, which can be found on the attached CD (see Appendix C).

3.4.1 Elementary melody

It was possible to create a playable MIDI file with basic melody after determination of values representing notes of each frame from the speech sample. Those were obtained by simple substitution of the pitch value into the formula listed in the Equation 2.2. One note per each 10 ms frame is played in this case.

The melody sounds quite fast and confusing because of this short duration of separate notes.

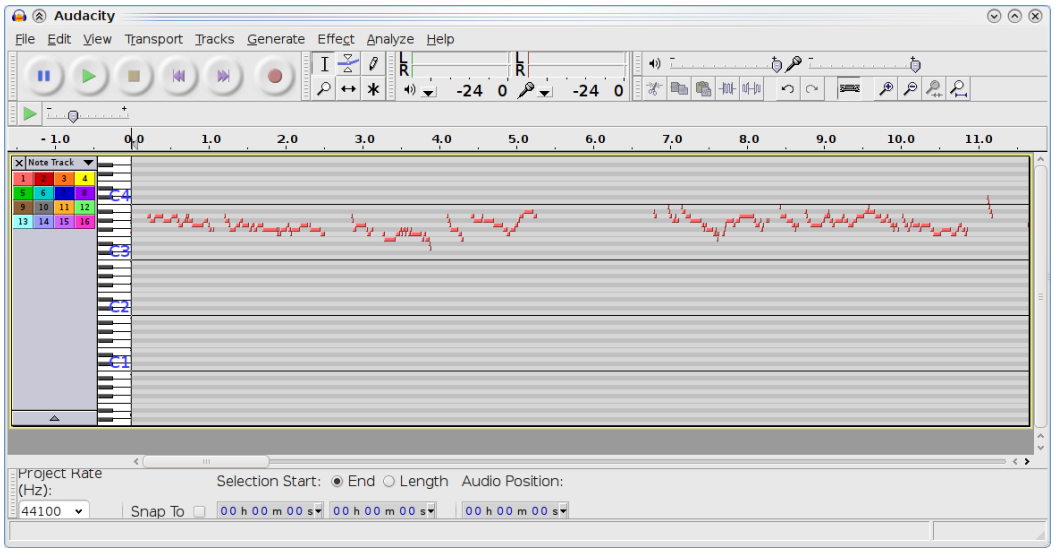


Figure 3.1: Visualization of MIDI file containing the elementary melody (note played on every 10 ms frame).

3.4.2 Averaging

A few algorithms were developed in order to improve the appearance of the melody and to make it more similar to the original speech sample. The basis of all of them was to calculate only one value representing the tone of e.g. 10/20/30 consecutive frames *(values chosen for testing purpose only - number of frames to average can be specified by user when running the application by specifying the correct command line option at the startup - see B.1)*. This was done by simply calculating the mathematical average of notes in 10/20/30 following frames.

The melody got more comfortable for listening. The reason is, that the tones are played for a longer time (10/20/30 ms) than in the elementary melody sample. But there is a

drawback in this approach. The tones do not fit the original tones interval of the melody in some cases anymore (e.g. when frames detected as pauses were included to the calculation of the mathematical average). Such tones were further cut off as described further.

An enhancement was made in the final version of the application. Tones are marked as pauses, if a pause occurs in at least one of the frames taken for the averaging. Tones not fitting the original tones interval do not occur any more and therefore the cutting off algorithm is not necessary any more.

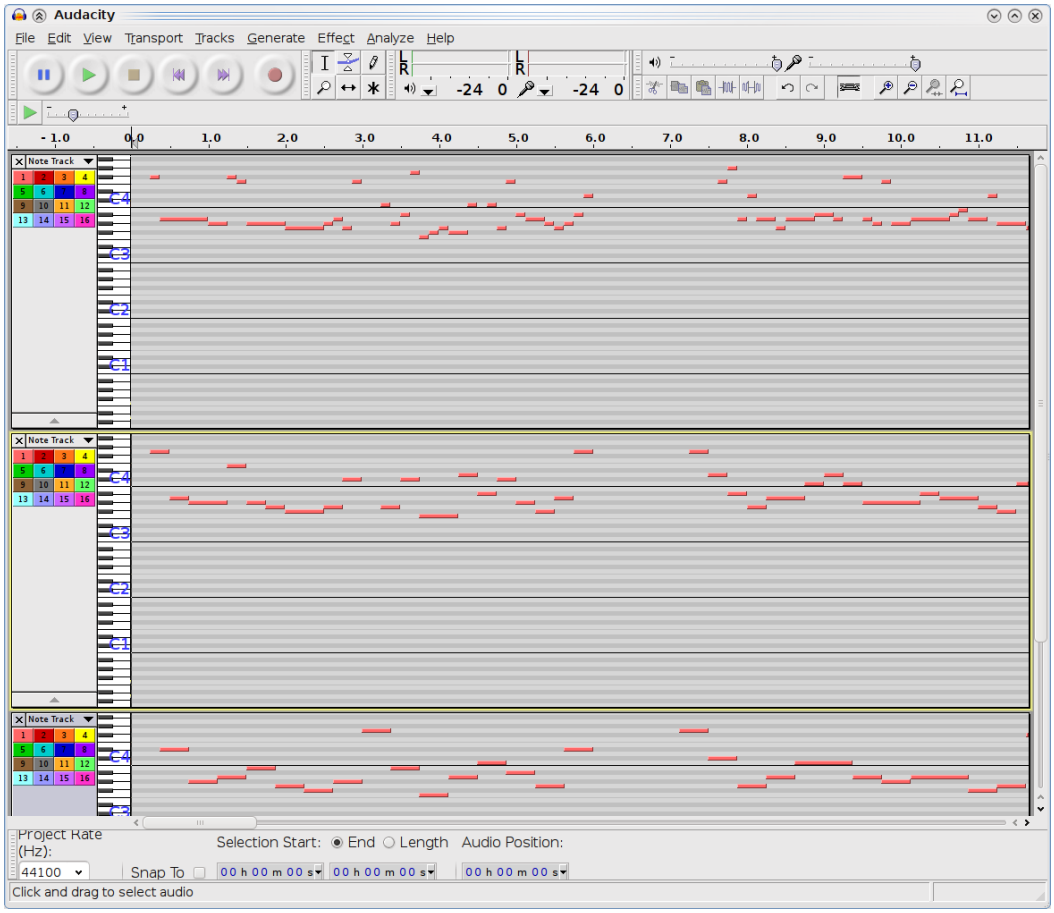


Figure 3.2: Visualization of MIDI files containing melody with tones calculated for 10/20/30 following frames (starting with the first line).

3.4.3 Cutting off

This enhancement was made in order to cut off those values not fitting the original tones interval. Therefore those clusters of frames were marked as pauses. This however did not have much positive impact on the resulting melody, so this function is not included in the final version of the application. Moreover another enhancement was made, so that situations of not fitting the original tones interval do not occur any more.

3.4.4 Merging

If consecutive clusters of frames represented the same note, they were merged and played as one note lasting for appropriate period of time.

The output melody got a bit closer to the original sample, the melody line started to evoke the line of melody of the original speech. Despite this the outcome was not satisfying yet. It was due to the 10 ms framing. It's very likely, that such short frames have a different fundamental frequency and therefore different tone, even if there are next to each other. Therefore the flow of melody seemed like it was very often „cut" by impedimentary tones. Therefore this function is not included in the final version of the application.

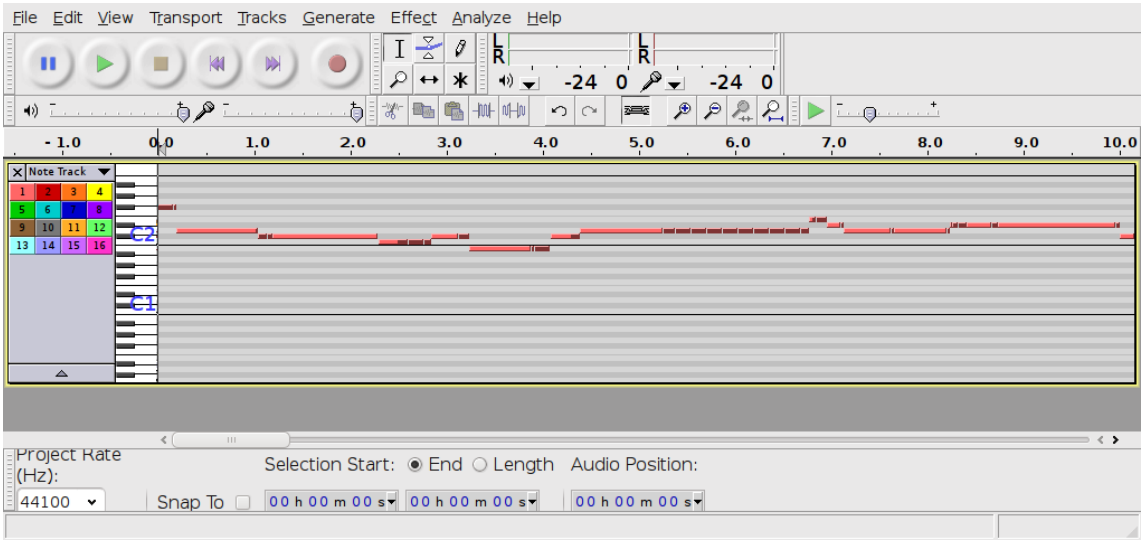


Figure 3.3: Visualization of MIDI file containing melody with tones merged.

3.5 Use of phoneme recognizer

One of our goals was to make the output melody as similar to the input sample as possible, therefore the „rhythm" (or rather timing of notes) has to be determined from something else than frames length. Therefore the phoneme recognizer(described in the Section 2.5) was used.

Analysis of speech using the phoneme recognizer provides information about each phoneme or pause or noise in the input speech. This is important, because we need to know when and for how long the phoneme lasts in order to make the timing of the notes in our output melody as accurate as possible. The phoneme analysis also tells us what kind of phoneme is the sound (whether it is a consonant or vowel).

3.6 Syllables detection

These data were used for further syllables detection in the input speech. The algorithm for syllables detection was developed. Its principle is very simple. The syllable was defined as a sequence of phonemes containing at least one vowel and ending with the first consonant

following the vowel. Values representing the note of each syllable were calculated similarly as mentioned in the Section 3.4.2.

The result output melody sounds very like the original melody of the input speech.

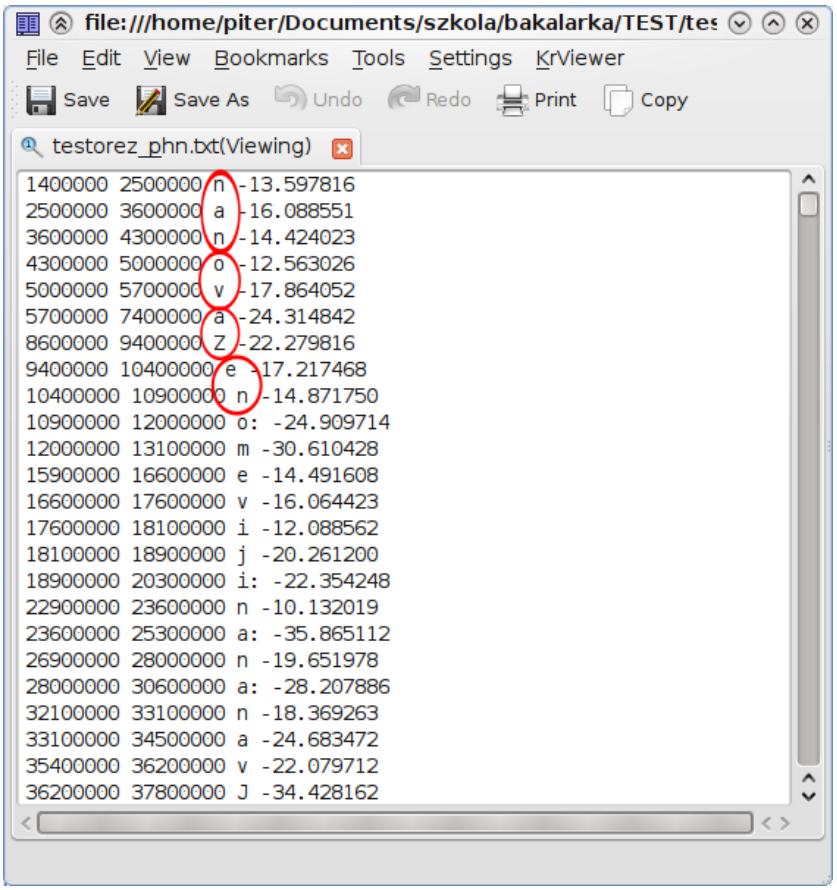


Figure 3.4: Sample phoneme recognizer output with detected syllables.

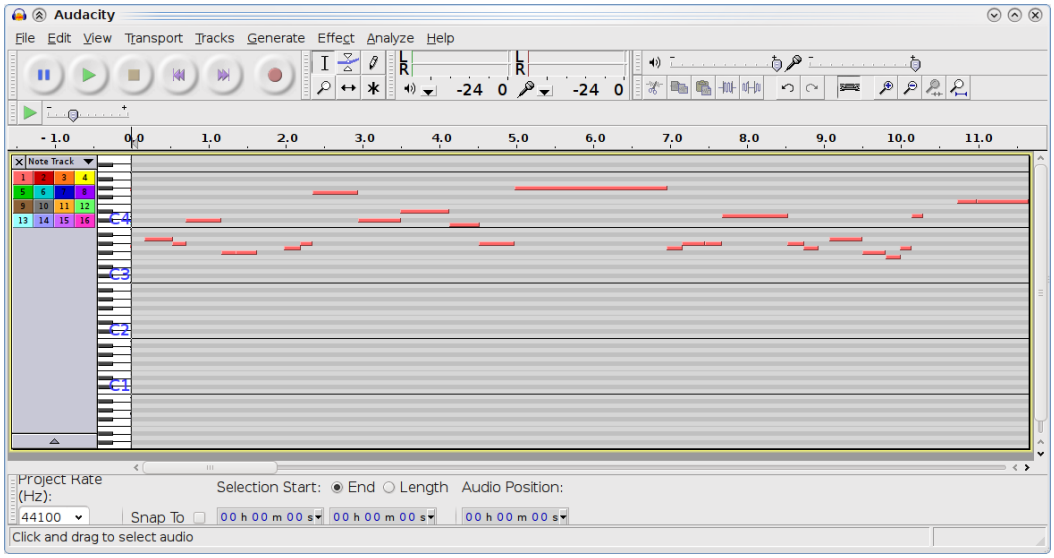


Figure 3.5: Visualization of MIDI file containing melody created by using the timing of detected syllables.

Chapter 4

Criteria of beauty in music generation

4.1 Motivation

It was very important to think of what is important in music generation, what makes a melody/composition more pleasant for a listener. There is also a lot of compositional methods, which can be used when creating more complex musical composition than just one-line melody. These musical data and compositional methods were discussed with contemporary composer Mgr. Ing. MgA. Dan Dlouhý, Ph.D. and the outcomes have been used in the application.

4.2 Dynamics

In music, dynamics refers to the loudness or softness of music (e.g. volume and accent changes) [7]. Dynamics was considered as a parameter of melody, that makes it more interesting. Alternation of dynamics during the melody play can evoke different feelings e.g. grading the tension, etc.

There were no dynamics included in the basic melody generated from speech as described in the previous chapter 3. Note values have been changing, however the volume of each note was on the same level throughout the whole melody.

4.3 Volume calculation

The power values obtained as described in the Section 2.4 were taken into consideration. Values representing the volume of each note were calculated out of the power values similarly to the averaging of the notes (see the Section 3.4.2). These values were further normalized by the ratio calculated as described in the Equation 4.1 in order to spread the values over the MIDI Velocity parameter values range. The resulting values are further used as the MIDI Velocity parameter.

$$k = \frac{Vel_{max}}{power_{max}} \tag{4.1}$$

Where k stands for the ratio, Vel_{max} stands for the maximum MIDI Velocity value = 127 [6] and power_{max} stands for the maximum out of the power values of all frames (see

the section 2.4).

The resulting melody reminds the input speech even more. The dynamics evoke a feeling of accent in words of the input speech thanks to the changing volume of each syllable.

4.4 Musical scales

In music, a scale (also musical key) is a group of musical notes collected in ascending and descending order, that is used to conveniently represent a musical work including melody and harmony [16]. It is considered to sound quite pleasant for a listener, if a melody is composed/played in one scale/key.

There are many different types of scales such as major, minor, chromatic, whole tone, pentatonic, etc. Only major and minor were used in the application. The user can define, which one will be used.

4.4.1 Key note

The 1st tone (tonic) of each scale is called the key note, which is further used for making the key signatures (not important in our application).

However, the most frequent tone in the melody is considered to be the key note of the scale in case of this application. Therefore the histogram of tones in the melody is created and it is further used for the most frequent tone detection (for histogram implementation see the Section 5.5).

4.4.2 Major scales

Melodies/compositions created/played in major scales usually sound happier than in the minor scales [1]. The major scale consists of seven different pitches. There are half steps (half tone) between the third and fourth and seventh and eighth scale degrees; whole steps (whole tone) exist between all other steps [11].

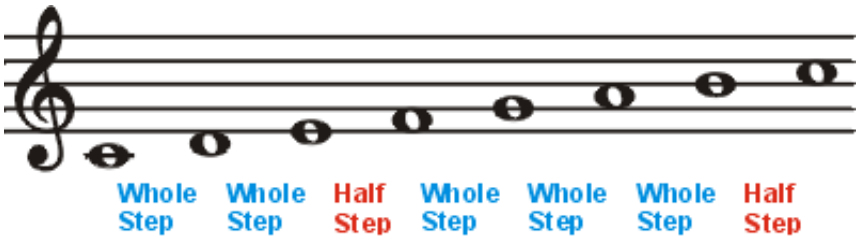


Figure 4.1: A C major scale (steps between the tones are marked) [11].

4.4.3 Minor scales

Melodies/compositions created/played in minor scales usually sound sad or scary [1]. These scales have seven different scale degrees. The natural minor scale was taken in the consideration. There are half steps between the second and third and the fifth and sixth degrees; whole steps exist between all other steps [11].

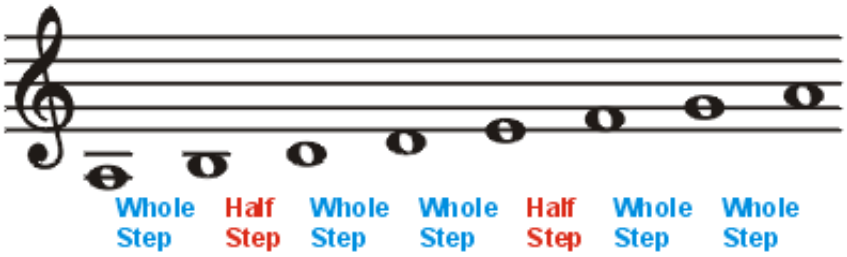


Figure 4.2: An A natural minor scale (steps between the tones are marked) [11].

4.4.4 Cutting off

The speech melodies obtained, as described in previous chapters don't seem to be in just one scale/key, that's why they sound in a way, which could be described as wild, confusing, strange or not making any sense. Therefore an algorithm was developed to determine tones not fitting the key/scale.

Every tone of the melody is checked, if it fits the scale (major or minor). Such tone is cut off, if it does not fit the scale. The pause takes place for the time of the original tone playtime.

4.4.5 Adding threshold

Even though the melody still sounds quite confusing after cutting off the notes not fitting the scale, it somehow also started to feel a bit boring as there are no interesting tones from the other scales. A computation of a random number may be added to the cutting off algorithm. Therefore a uniform distribution random number from the interval of 0 to 1 may be generated and checked against the threshold. The threshold was set to 0.8. Therefore the tone will not be cut off/will be played even if it does not fit the scale, if the generated random number is greater than the threshold (see the pseudo-code 4.3).

```
if (uniform(0, 1) > 0.8 and not(tone_is_in_scale), scale_key, scale_type):
  pause(playtime_of_the_tone)
else:
  tone_is_played
```

Figure 4.3: Pseudo-code illustrating the cutting off of the tones not fitting the scale of melody with added threshold. The scale_key is detected as described in the Section 4.4.1 and the scale_type is specified by the user as described in Appendix B.

4.5 Chords

In this application a chord stands for a triad of tones. Playing a whole chord instead of just one tone in the melody at a time makes the result more interesting for the listener.

These triads can be major or minor depending if the melody is in major or minor scale. A triad is a chord that contains 3 notes [17]: Tonic (1st tone) + 4th/5th halftone (minor/major scale) + 8th halftone.

Chords can be played over every single note in the melody or over every Nth note (can be configured by setting the command line option for the application).

4.6 Compositional Methods

There is a lot of different compositional methods, which are used by composers, when post-processing a melody or creating compositions based on a melody theme.

4.6.1 Inversion of notes

Intervals, chords or melodies can be inverted in music.

Inversion in this application stands for inverting each tone in reference to the Concert A. This means that if a tone is e.g. 14 halftones below the Concert A it is inverted to be lying 14 halftones above the Concert A.

4.6.2 Changes of tempo

The tempo of the melody can be changed easily by multiplying the length of the original tone timing by a user-defined constant.

4.6.3 Adding a bass line

Some musicians say, that when the bass is played in a song, nothing happens, but if it is not there, you miss it. Therefore adding the bass line is considered very important in order to highlight the character of the melody.

Adding the bass line was made by simply adding to the melody the same tone as the original tone in the melody, but shifted two octaves (24 halftones) lower, when the chords are played.

4.7 Use of more than one method/aspect at the same time

All of the above methods and algorithms mentioned above were implemented in the application and can be used separately or all together up to the users will. The user can define which of them to use by setting the correct command line option when running the application.

4.8 Different musical instruments

Playing the melody on different instruments also makes the listening of the music (it is not only just one line of the melody any more, if we join together few of the techniques mentioned earlier) both more interesting and easier to listen. It somehow feels like the music is played by a band. However, the quality of such a feeling depends on the MIDI instrument we choose. Those, that are built in the system by default, usually sound very „childish“, but there is plenty of professional MIDI virtual instruments, which sound very similar to the real ones.

Chapter 5

Implementation

5.1 What user wants to do

The user specifies use of the individual functions by adding the correct command line options at the program startup (see manual in Appendix B). These parameters are processed by standard Python (Section A.1) function getopt(). The array of options and their arguments is returned. The array is checked for required options and eventually their arguments, if the argument is mandatory. The arguments are also checked, if they are correct.

5.2 Obtaining the basic information from given WAV file

The analysis of the input speech to be converted can start after the filename of the file containing the speech is extracted from the command line arguments specified at the application startup. Functions from the the Snack toolkit (see the subsection A.2.1) for pitch and power data calculation are run with the parameters set to the default values (length of the frame set to 10 ms, etc.), if the input WAV file was opened successfully. Output returned by these functions is processed and stored in separate text files for later use.

Values representing tones in each frame are calculated using the fundamental frequency values (pitch) obtained earlier. Tones are represented as a distance from the Concert A tone measured in halftones (as described in the Section 2.2). If the tone result value representing the tone is 0, such tone is further checked. The „pau“ mark is added to the result, if the pitch value is 0 Hz in the particular frame. These values are stored into another text file for later use.

5.3 Averaging

Averaging takes place in cases, where the frames number was specified as the parameter of the application. The averaging function works with both text files containing power and tone values. Power is used for the volume calculation as described in the Section 4.3. Values representing tones are used for the calculation of the exact note value.

Text file (filename containing the number of frames to be averaged as one note) is the result of the averaging function. It contains the timing, note value and volume of one note on each row. A simple mathematical average is calculated from those values. Notes are joined together if two same notes occurred next to each other. The „pau“ mark is added, if a pause should be played. Fragment of the output text file is shown in the Figure 5.1.

5.4 Syllables detection

Syllables detection works very similarly to the averaging. The text file containing the output of the phoneme recognizer is used. The file was earlier processed to not contain non-speech events (speaker noise, pauses, etc.). This does not mean any loss of data important for the syllables detection, as the exact timing is specified for each phoneme. The syllables are detected as described in the Section 3.5 based on the data from the text file. The timing of the syllables is also determined from this text file. The exact note and its volume is calculated similarly to the averaging (see the Section 5.3) as the timing (start time and duration) of the syllables is now known. Free spaces between the syllables are marked as pauses („pau“).

Text resulting text file is in the same format as the text file created by averaging (see the Section 5.3). Fragment of the output text file is shown in the Figure 5.1.

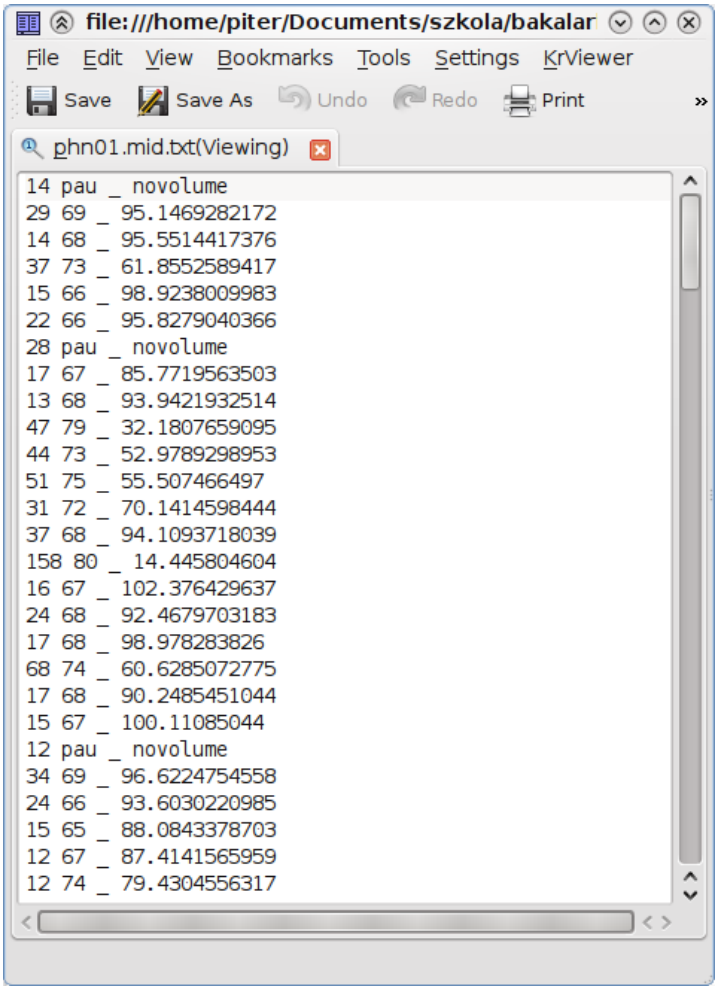


Figure 5.1: Fragment of a sample output text file further used for the creation of the resulting MIDI file. This particular fragment is a result of the syllable detection (see 5.4.

5.5 Scale key note detection

User may require to cut off tones in the melody, which are not fitting the scale key of the melody. Therefore the scale key of the melody needs to be determined first (as described in the Section 4.4.1). The text file obtained as described earlier is scanned and an array, where an index stands for the tone and value on each index stands for the number of occurrences of the tone. The array is scanned and the index, where the value is the highest is declared as the scale key note.

5.6 Determining if the tone fits the scale

Every tone is checked to fit the scale of the melody, if the application user requires it. The type of the scale (major or minor) has to be specified as input parameter. The correct pattern of intervals between single notes of the major/minor scale (described in the Section 4.4.2 or 4.4.3) is taken into consideration.

5.7 Creation of the MIDI File

At first, the final filename of new MIDI file is created by adding the shortcuts representing single command line parameters specified by the user. Text file containing the output of averaging or syllable detection functions is taken as input. This file contains all the information needed for further creation of the MIDI file. The basic MIDI melody (without any enhancements chosen and specified by the application parameters) can be created directly from the data in this text file by creating the Note on and Note off event messages (see the Section 2.6) in the timing described in the given input text file.

The user-specified options e.g. cutting off (it the tone does not fit the scale), scale type, tempo adjustment are taken into consideration.

Tempo The timing of the notes is multiplied by the tempo adjustment ratio specified by the user at the application startup.

Volume The Velocity parameter is set to the volume value from the input text file normalized by the ratio (see the Section 4.3) by the creation of the Note on event message, if the volume should be taken into consideration as user requested. Otherwise the default Velocity value is set to 64 (the middle of the possible 0 to 127 range) [6].

Cutting off with added threshold The scale key note is determined at first as described in the Section 5.5. Each tone/note of the melody can be checked, if it fits the scale as the scale key and scale type (specified by user at the startup) is known now. The principle is described in the Sections 4.4.4 and 4.4.5.

Chords + the bass line The type of the scale has to be set by user at the startup in order to create chords. Chords stand for triads in this application. Two tones are added to the original one:

- 4th and 7th halftone (counted from the original tone) for the major scale
- 3rd and 7th halftone (counted from the original tone) for the minor scale.

The bass line is created by simply adding new tone parallel to the melody. The tone is 24 halftones below the original one (same tone, but played two octaves lower).

Inversion New tone is added to the original one in case of inversion. The new tone is inverted in reference to the Concert A as described in the Section 4.6.1.

Musical instruments Different melody lines are assigned to be played on different musical instruments:

No. 21 Reed Organ The original melody and chords are played on the reed organ.

No. 34 Electric Bass (finger) The bass line added to the chords is played on the electric bass.

No. 67 Tenor Sax The inverted tones of the melody are played on the tenor saxophone.

The instruments were chosen out of the default midi instrument patch map [3].

The result MIDI created by adding some of the options required by the user as listed in this section or in the Table B.1 can be seen in the Figures 5.2, 5.3, 5.4.

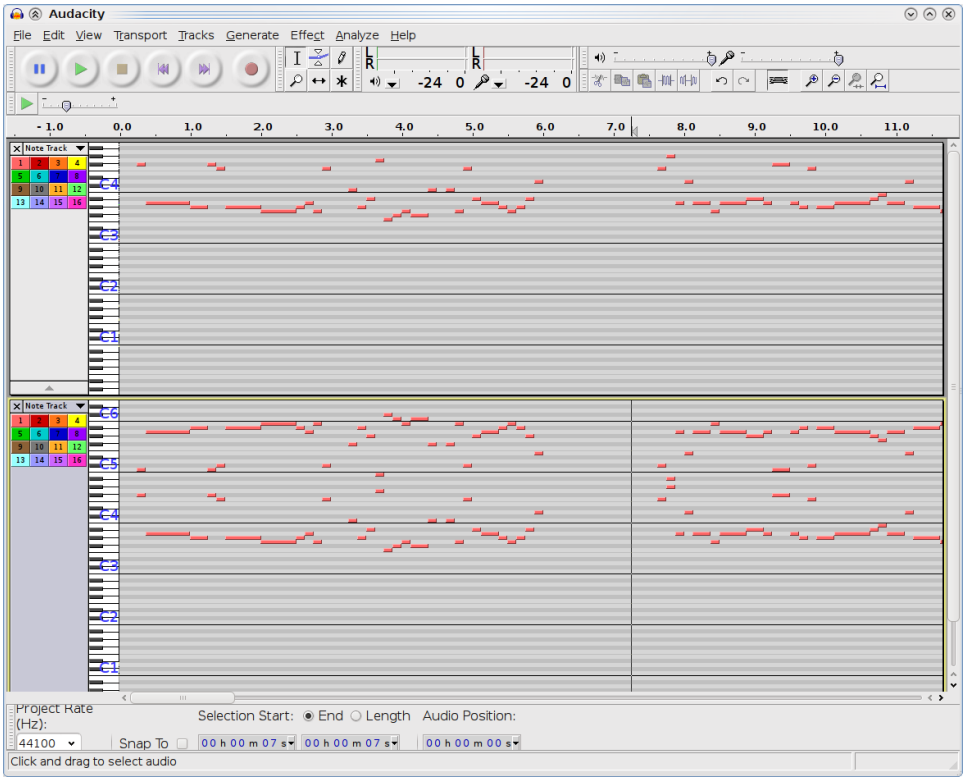


Figure 5.2: MIDI melody averaged for every 10 ms frames (top) presented together with the same melody with inverted line added (bottom).

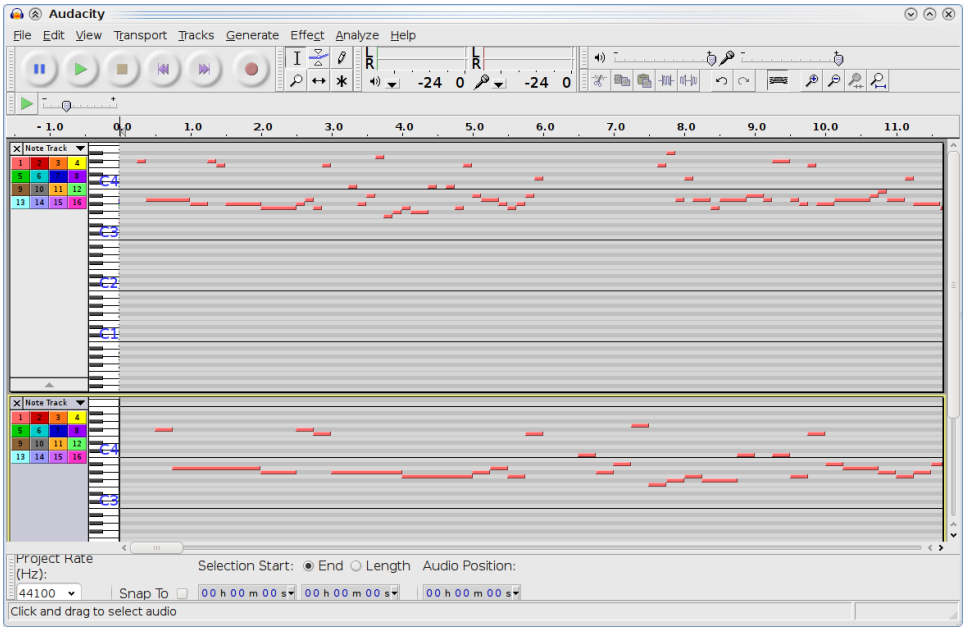


Figure 5.3: MIDI melody averaged for every 10 ms frames (top) presented together with the same melody with a tempo ratio set to 2.0 (bottom).

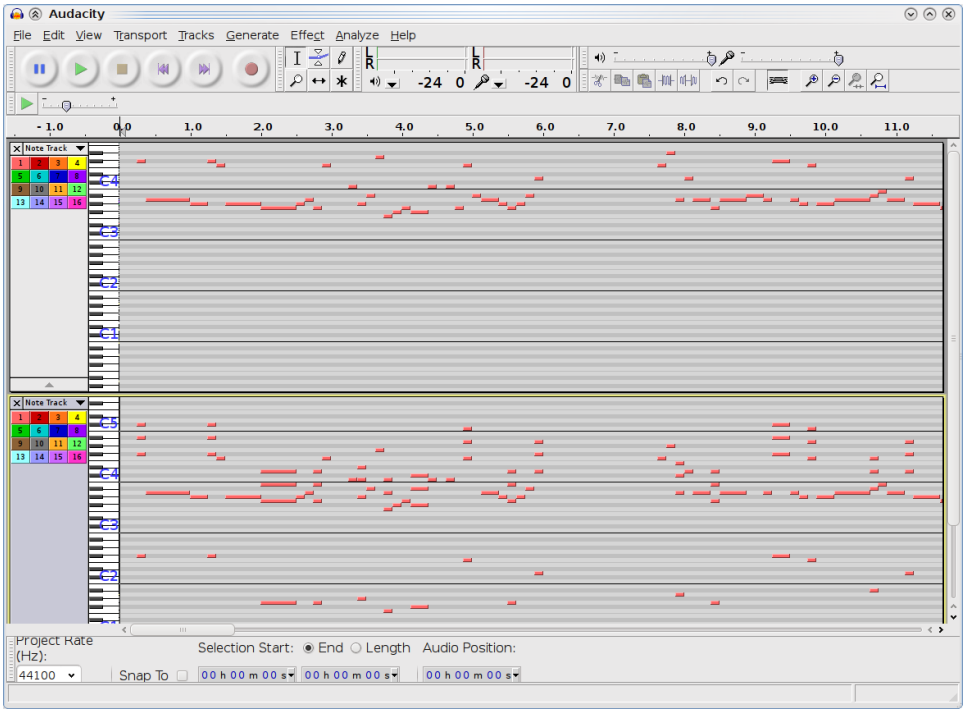


Figure 5.4: MIDI melody averaged for every 10 ms frames (top) presented together with the same melody with major chords (and the bass line) played on every 3rd note and notes not fitting the major scale cut off (bottom).

5.8 Waveform display

Displaying the waveform of the input speech is an optional feature. User can require it by setting the corresponding command line option (as described in the Table B.1). The display of the waveform is handled by the tkSnack function (see the AppendixA.2.1). Sample waveform can be seen in the Figure 5.5.

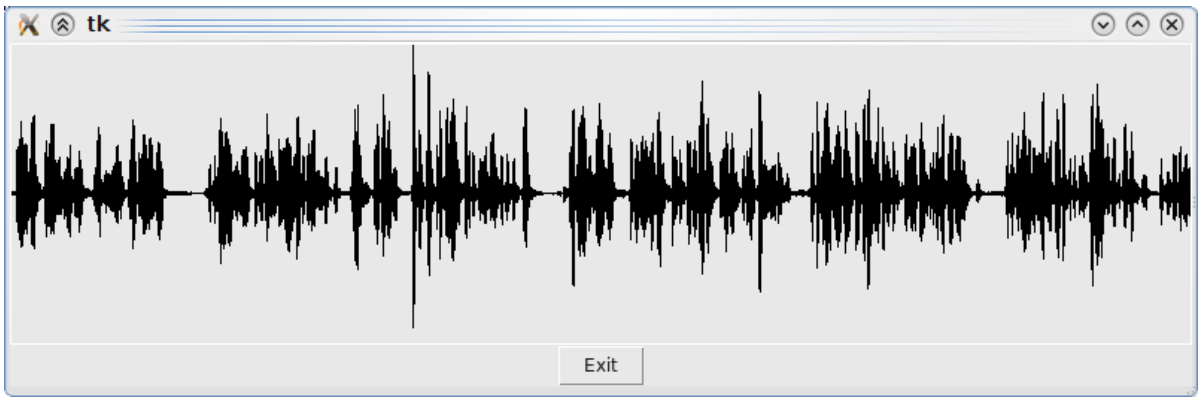


Figure 5.5: Fragment of sample waveform displayed by the application.

Chapter 6

Results and evaluation

6.1 Examples and user Testing

A Few examples were created to find out how different people like the music generated by the application. The examples as well as the corresponding input speech file can be found on the enclosed CD as described in Appendix C. Different combinations of techniques were used to create these examples in order to find out, which of the methods people like the most. Methods used in a particular sample can be detected according to the suffix of the MIDI file (also described in Appendix B.1):

20frames_chordsEvery2_minor_inv_1.0t.mid Tones averaged for every 20 frames, minor chords played on every 2nd note, inverted line added, original tempo.

20frames_chordsEvery5_major_cut_0.8t.mid Tones averaged for every 20 frames, major chords played on every 5th note, out of scale notes cut, tempo ratio 0.8.

20frames_chordsEvery5_major_cut_2.2t.mid Tones averaged for every 20 frames, major chords played on every 5th note, out of scale notes cut, tempo ratio 2.2.

30frames_1.0t.mid Tones averaged for every 30 frames, original tempo

5frames_vol_1.0t.mid Tones averaged for every 5 frames, volume taken into count, original tempo

phn-out_chordsEvery2_minor_inv_1.0t.mid Tones averaged based on syllables detection, volume taken into count, minor chords played on every 2nd note, inverted line added, original tempo

phn-out_chordsEvery3_major_1.0t.mid Tones averaged based on syllables detection, major chords played on every 3rd note, original tempo

phn-out_vol_chordsEvery2_minor_inv_1.0t.mid Tones averaged based on syllables detection, minor chords played on every 2nd note, inverted line added, original tempo

The input speech file was played to the testing person first. Then the resulting MIDI files were played. Each person then had to grade each result music by giving a mark from 1 to 5 (5 = the best, 1 = the worst).

6.2 Testing results and evaluation

Sample	**Test person 1**	**Test person 2**	**Test person 3**	**Test person 4**	**Average**
20frames _chordsEvery2_minor _inv_1.0t.mid	3	1	3	1	2
20frames _chordsEvery5_major _cut_0.8t.mid	5	5	4	5	4.75
20frames _chordsEvery5_major_cut _2.2t.mid	4	1	2	4	2.75
30frames_1.0t.mid	2	1	2	3	2
5frames_vol_1.0t.mid	4	4	4	2	3.5
phn-out_chordsEvery2 _minor_inv_1.0t.mid	4	2	2	1	2.25
phn-out_chordsEvery3 _major_1.0t.mid	5	3	5	2	3.75
phn- out_vol_chordsEvery2 _minor_inv_1.0t.mid	4	2	3	1	2.5

Table 6.1: Grades given by testing people for each example (5 = the best, 1 = the worst).

Few interesting facts can be drawn when comparing the results. Adding the inverted line does not seem to have a good influence on the resulting beauty of the music. Adding volume changes does not seem to have much influence as well as using the syllable detection based on the phoneme recognizer output instead of number of frames for averaging. Smaller number of frames taken for averaging seems to improve the beauty of the music as well as keeping original or choosing even faster timing (lower tempo ratio). Adding chords with the corresponding bass line seems to improve the beauty of music, however it does not seem to matter what the timing of the chords is.

Chapter 7

Conclusion and future work

7.1 Summary

The resulting application extracts pitch and power data from the WAV file containing the input speech. Values representing tones are calculated based on the pitch data. Volume values may be calculated based on the power data. The syllables detection is performed if the file containing the phoneme recognizer output is specified. Further the tones are averaged as required by the user. User specified post-processing (e.g. inversion of tones, adding the chords, etc.) is performed further. The resulting music is stored as a MIDI file, whose filename describes which techniques were used to create the music.

The results may be considered as satisfactory. The resulting music sounds a bit confusing as was expected considering that it is a speech-melody-based music. Melodic similarity of the resulting music to the original speech melody may be hearable (especially, when the phoneme recognizer output is used for syllables detection - therefore better timing of the notes is achieved, etc.).

7.2 Contribution of the thesis

Discovering the different compositional methods and discussing them with Mr. Dlouhý was very inspiring for me as a developer of the application and as an amateur musician. I had to refresh my knowledge about musical scales which I learned at the art school.

I have learned the basics of the LaTeX and the Python programming language. I have also learned some interesting tools as listed in Appendix A.

7.3 Possible further use of the application

Contemporary composers use many different sources to gain inspiration and obtain data, which can be used further in their music compositions. If things like motion or light sensors, bitmap pictures, fractal functions, etc. can be used to obtain this kind of data, than why not the human speech? Therefore the application could probably be used mostly by the todays music composers. The output music may be directly played or (even more importantly) it may be used as an idea tank for the composers searching for new melody/music themes for their compositions.

Mr. Dlouhý has already expressed an interest in using this application, as he may further re-factor the results in his compositional work.

Possible educational use was also discussed with Mr. Dlouhý. He expressed an idea, that the application could be used to demonstrate different compositional techniques to the composition students. However, for this use, the tool should be completed by functionalities described in the Section 7.4.1 probably would have to be implemented.

7.4 Future work

The future direction of this application development has to be discussed. There are two different ways, which appear to be contradictory. The result music could be as similar as possible to the original input speech or it could be a harmonized heavily post-processed speech-melody-based music.

7.4.1 Music composition

There are many different compositional methods, which could be implemented in the future, on the top of those already implemented in the application. Here are some examples:

Tempo differing The bass line could be played in a different tempo, than the melody theme, etc. e.g. the bass line would be played slower then the melody theme and some higher tones played even „faster“ would be added to the melody.

Tones overlaying The tones in the melody could overlay one another. This would cause the music to feel more flowing.

Moreover, different MIDI library may also be considered. The Python MIDI Package used in this project does not implement some functions such as sustain, etc., which would be useful when implementing the features listed above. Satisfactory documentation is also hard to find for the Python MIDI Package.

The application may be further linked to some harmonization program, which would be trained to harmonize according to different music styles e.g. jazz, renaissance, minimalism, etc.

A function for notation creation may be added. However, there are still some challenges, which have to be faced. The biggest of them is probably to define, how to extract the correct time signature. This is particularly hard, when the syllables are considered as the source to generate note timing.

7.4.2 Resemblance to the input speech

Exact pitch values may be used instead of tone calculation and rounding. This approach is known as the micro-interval tuning. However, some problems could occur when creating the MIDI files. Probably the pitch bending technique would have to be used.

It is also possible, that the speech is spoken/„played“ in a whole new scale (not major, minor, pentatonic, etc.). Frequency analysis would be necessary to find out and define such a new scale.

Bibliography

[1] All About Key Signatures. [online], [cit. 2010-05-17].
URL http://www.empire.k12.ca.us/capistrano/mike/capmusic/key%20signatures/key_signatures.htm

[2] Czech SpeechDat-E sample page. online, [cit. 2010-05-17].
URL http://www.fee.vutbr.cz/SPEECHDAT-E/sample/czech.html

[3] General MIDI Level 1 Sound Set. [online], [cit. 2010-05-17].
URL http://www.midi.org/techspecs/gm1sound.php

[4] Glossary (Section MIDI). [online], [cit. 2010-05-17].
URL http://en.flossmanuals.net/Ardour/Glossary

[5] Gwary polskie. Przewodnik multimedialny. [online], [cit. 2010-05-17].
URL http://www.gwarypolskie.uw.edu.pl/

[6] MIDI Messages. [online], [cit. 2010-05-17].
URL http://www.midi.org/techspecs/midimessages.php

[7] Music Definition. [online], [cit. 2010-05-17].
URL http://www.enjoythemusic.com/musicdefinition.htm

[8] Phoneme recognizer based on long temporal context. [online], [cit. 2010-05-17].
URL http://speech.fit.vutbr.cz/en/software/phoneme-recognizer-based-long-temporal-context

[9] Python Midi Package. [online], [cit. 2010-05-17].
URL http://www.mxm.dk/products/public/pythonmidi

[10] Python Programming Language. [online], [cit. 2010-05-17].
URL http://python.org/

[11] Scales and Key Signatures. [online], [cit. 2010-05-17].
URL http://method-behind-the-music.com/theory/scalesandkeys

[12] The Snack Sound Toolkit. [online], [cit. 2010-05-17].
URL http://www.speech.kth.se/snack/

[13] TkInter. [online], [cit. 2010-05-17].
URL http://wiki.python.org/moin/TkInter

[14] WAVE Audio File Format. [online], [cit. 2010-05-17].
URL http://www.digitalpreservation.gov/formats/fdd/fdd000001.shtml

[15] What's Concert Pitch? [online], [cit. 2010-05-17].
URL http://www.concertpitchpiano.com/WhatsConcertPitch.html

[16] Benward, B.: *Music: In Theory and Practice : Spiral*. Mcgraw-Hill College, 2003, iSBN 978-0072942620.

[17] Murphy, D. B.: Triads.

[18] Schwarz, P.: *Phoneme recognition based on long temporal context*. Dizertační práce, 2009.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9132

[19] Schwarz, P.; Matějka, P.; Černocký, J.: Towards Lower Error Rates In Phoneme Recognition. *Lecture Notes in Computer Science*, ročník 2004, č. 3206, 2004: s. 465–472, ISSN 0302-9743.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=7483

[20] Schwarz, P.; Matějka, P.; Černocký, J.: Hierarchical structures of neural networks for phoneme recognition. In *Proceedings of ICASSP 2006*, 2006, s. 325–328.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8134

[21] Ying, G. S.; Jamieson, L. H.; Michell, C. D.: A Probabilistic Approach to AMDF Pitch Detection. In *Proceedings of the 1996 International Conference on Spoken Language Processing*, s. 1201–1204.

[22] Černocký, J.: *Zpracování řečových signálů - studijní opora*. Speech@FIT, Ústav počítačové grafiky a multimédií, Fakulta informačních technologií, Vysoké učení technické v Brně, 2006.

Appendix A

Cook book

A.1 Programming language

The application was implemented in Python programming language [10]. It was chosen because of availability of tools, that were easy to use with this programming language. The Python download files and documentation can be found on www.python.org.

A.2 Used tools

A.2.1 Tkinter + Snack

You need to use Tkinter [13] in order to use the graphical functions of the Snack library [12]. The Snack library provides functions needed for speech analysis. Therefore it was used for obtaining the fundamental frequencies, power, etc from the speech sample. Functions for visualizations of speech parameters (like waveform, etc.) are also provided in this library.

A.2.2 Phoneme recognizer based on long temporal context

The data for syllables detection were obtained by the phoneme analysis of the input speech. For this analysis the Phoneme recognizer based on long temporal context was used. The phoneme recognizer [8] was developed at Brno University of Technology, Faculty of Information Technology. The input speech needed to be re-sampled to 8kHz sample rate in order to be analyzed by the phoneme recognizer. The following parameters were used, while running the analysis:

```
phnrec -c PHN_CZ_SPDAT_LCRC_N1500 -i INPUT.wav -o OUTPUT.rec
```

The Czech recognition system was set (PHN_CZ_SPDAT_LCRC_N1500). The output was considered sufficient enough, even though test input samples were spoken in polish dialect.

A.2.3 Python Midi Package

The Python Midi package [9] is a collection of classes handling MIDI input and output in the Python programming language. The same package was used e.g. by creators of the Frets On Fire game.

Appendix B

Manual for the application

B.1 Requirements

The application was developed and tested to be run under the LINUX operating system. The Python programming language has to be supported. Tools listed in the Appendix A are also required and can be found together with the program on the enclosed CD.

The Brno University of Technology phoneme recognizer has to be installed in order to use the syllables detection data for the tones averaging. The installation package can be found on the enclosed CD.

B.2 User specified options

Application is run by typing the `./program.py` into the system console. User defines the resulting melody features by setting the corresponding command line options (as described in the Table B.1.

The filenames/filename suffixes of the result MIDI files represent the actions and features specified by the user as the command line options. The suffixes are described in the B.2.

Option	Description	Mandatory
`-c NUMBER`	Adds chords to the result music. Chords are played on every NUMBERth note. The bass line (tone) is added to the result music together with the chord.	NO
`-f FILE`	Specifies the input WAV file containing speech.	YES
`-F NUMBER`	Specifies number of frames taken for tone averaging.	YES (if the `-p FILE` not set)
`-h`	Prints application help information.	NO
`-i`	Adds inverted line to the resulting music.	NO
`-p FILE`	Specifies the file containing the phoneme recognizer output corresponding with the input WAV file.	YES (if the `-F NUMBER` not set)
`-s [major|minor]`	Specifies the scale type, which is taken into consideration, when adding the chords or cutting the tones not fitting the scale of the melody.	YES (if the `-c NUMBER` or `-S` is set)
`-S`	Cuts notes, that do not fit the melody scale (according to the specified scale type).	NO
`-t RATIO`	Specifies the tempo RATIO (must be a floating point number). The default ratio is 1.0 (original timing) if the option is not set.	NO
`-v`	Adds volume changes to the result music.	NO
`-w`	Shows waveform of the input WAV file.	NO

Table B.1: Application command line options.

Suffix	Description
`.mid`	Result MIDI file.
`.mid.txt`	Text representation of the MIDI file.
`Xframes`	X frames were taken for averaging.
`_vol`	Volume changes added to the result music.
`_chordsEveryX`	Chords and bass tone played on every Xth tone.
`_major`	The scale type set to major.
`_minor`	The scale type set to minor.
`_inv`	Inverted line added to the music.
`_Xt`	Tempo ratio set to X.
`FILE`	The FILE was used for the syllables detection data taken for averaging

Table B.2: Output filenames/filename suffixes table.

Appendix C

CD contents

A CD containing the executable source code as well as the technical report (together with LaTeXsource codes) of this work belongs to the thesis. Libraries as well as the needed tools are also placed on the CD. However, they need to be installed first as described in the corresponding readme file.

Examples of the input data as well as sample resulting MIDI files are also attached on the CD.